Métodos de ordenamiento

Descripción

Este código realizado en lenguaje C implementa diversas estructuras de datos (pilas, colas, listas, árboles) y un método de ordenamiento (ordenamiento por inserción) para gestionar información de personas. El programa presenta un menú principal con opciones para operar con cada estructura y el método de ordenamiento.

Estructuras de Datos Implementadas

1. Pilas
2. Colas
3. Listas
4. Árboles

Método de Ordenamiento Implementado

- Ordenamiento por Inserción

Funciones Auxiliares

- **Pilas**
  - `inicializarPila`
  - `push`
  - `pop`
  - `mostrarPila`
  - `modificarPila`
- **Colas**
  - `inicializarCola`
  - `encolar`
  - `desencolar`
  - `mostrarCola`
- **Listas**
  - `inicializarLista`
  - `insertarAlInicio`
  - `mostrarLista`
  - `eliminarLista`
- **Árboles**
  - `crearNodoArbol`
  - `insertarEnArbol`
  - `mostrarArbolInOrden`
- **Ordenamiento**
  - `ordenamientoInsercion`
  - `mostrarArreglo`

Funciones Principales

- `menuPilas`
- `menuColas`
- `menuListas`
- `menuArboles`
- `menuOrdenamientoInsercion`
- `main`

Detalles de las Estructuras de Datos

1. Pilas

La estructura de pila se define con un nodo que contiene información sobre una persona y un puntero al siguiente nodo. Las funciones permiten realizar operaciones en la pila.

Estructura de la Pila:

```
struct Pila {
    struct Nodo* tope;
};
```

La estructura `Pila` tiene un único miembro llamado `tope`, que es un puntero a un nodo. Cada nodo contiene información sobre una persona (`struct Persona`) y un puntero al siguiente nodo en la pila.

```
struct Nodo {
    struct Persona datos;
    struct Nodo* siguiente;
};
```

Funciones Relacionadas con Pilas:

- `inicializarPila`: Inicializa una pila vacía estableciendo el puntero `tope` en `NULL`.

```
void inicializarPila(struct Pila* pila) {
    pila->tope = NULL;
```

```
}
```

- `push`: Agrega una nueva persona a la pila.

```
void push(struct Pila* pila, struct Persona datos) {
    struct Nodo* nuevoNodo = (struct Nodo*)malloc(sizeof(stru
ct Nodo));
    // ... (verificación de la asignación de memoria)
    nuevoNodo->datos = datos;
    nuevoNodo->siguiente = pila->tope;
    pila->tope = nuevoNodo;
}
```

- `pop`: Elimina la persona en la cima de la pila.

```
void pop(struct Pila* pila) {
    if (pila->tope == NULL) {
        printf("La pila está vacía. No se puede elimina
r.\\n");
        return;
    }
    struct Nodo* temp = pila->tope;
    pila->tope = temp->siguiente;
    free(temp);
}
```

- `mostrarPila`: Imprime en la consola el contenido de la pila.

```
void mostrarPila(struct Pila pila) {
    if (pila.tope == NULL) {
        printf("La pila está vacía.\\n");
        return;
    }
    printf("Contenido de la pila:\\n");
    struct Nodo* actual = pila.tope;
```

```
    while (actual != NULL) {
        // ... (imprimir los datos de la persona en el nodo)
        actual = actual->siguiente;
    }
}
```

- `modificarPila`: Permite al usuario modificar la información de una persona en la pila.

```
void modificarPila(struct Pila* pila) {
    // ... (verificar si la persona a modificar está en la pi
la y realizar la modificación)
}
```

Menú de Pilas (`menuPilas`):
El menú específico para las pilas (`menuPilas`) utiliza estas funciones para realizar operaciones como agregar, modificar, eliminar y mostrar personas en la pila. Los mensajes en el menú guían al usuario a través de estas operaciones.

2. Colas

Similar a las pilas, la estructura de cola también se define con un nodo y un puntero al siguiente nodo. Las funciones facilitan las operaciones en la cola.

Estructura de la Cola:

```
struct Cola {
    struct Nodo* frente;
    struct Nodo* final;
};
```

La estructura `Cola` consta de dos punteros, `frente` y `final`, que indican el principio y el final de la cola, respectivamente.

Funciones Relacionadas con Colas:

- `inicializarCola`: Inicializa una cola vacía.

```
void inicializarCola(struct Cola* cola) {
    cola->frente = NULL;
    cola->final = NULL;
}
```

- `encolar`: Agrega una nueva persona al final de la cola.

```
void encolar(struct Cola* cola, struct Persona datos) {
    struct Nodo* nuevoNodo = (struct Nodo*)malloc(sizeof(stru
ct Nodo));
    // ... (verificación de la asignación de memoria)
    nuevoNodo->datos = datos;
    nuevoNodo->siguiente = NULL;
    if (cola->final == NULL) {
        cola->frente = nuevoNodo;
        cola->final = nuevoNodo;
    } else {
        cola->final->siguiente = nuevoNodo;
        cola->final = nuevoNodo;
    }
}
```

- `desencolar`: Elimina la persona en el frente de la cola.

```
void desencolar(struct Cola* cola) {
    if (cola->frente == NULL) {
        printf("La cola está vacía. No se puede desencola
r.\\n");
        return;
    }
    struct Nodo* temp = cola->frente;
    cola->frente = temp->siguiente;
    if (cola->frente == NULL) {
        cola->final = NULL;
    }
```

```
    free(temp);
}
```

- `mostrarCola`: Imprime el contenido de la cola.

```
void mostrarCola(struct Cola cola) {
    if (cola.frente == NULL) {
        printf("La cola está vacía.\\n");
        return;
    }
    printf("Contenido de la cola:\\n");
    struct Nodo* actual = cola.frente;
    while (actual != NULL) {
        // ... (imprimir los datos de la persona en el nodo)
        actual = actual->siguiente;
    }
}
```

Menú de Colas (`menuColas`):
El menú para las colas utiliza estas funciones para operaciones como agregar, eliminar y mostrar personas en la cola. Los mensajes guían al usuario a través de estas operaciones.

3. Listas

La lista enlazada se implementa con nodos que contienen información de personas y un puntero al siguiente nodo. Operaciones básicas están disponibles.

Estructura de la Lista:

```
struct Lista {
    struct Nodo* inicio;
};
```

La estructura `Lista` contiene un puntero `inicio` que señala al primer nodo de la lista.

Funciones Relacionadas con Listas:

- `inicializarLista`: Inicializa una lista vacía.

```
void inicializarLista(struct Lista* lista) {
    lista->inicio = NULL;
}
```

- `insertarAlInicio`: Agrega una nueva persona al inicio de la lista.

```
void insertarAlInicio(struct Lista* lista, struct Persona dat
os) {
    struct Nodo* nuevoNodo = (struct Nodo*)malloc(sizeof(stru
ct Nodo));
    // ... (verificación de la asignación de memoria)
    nuevoNodo->datos = datos;
    nuevoNodo->siguiente = lista->inicio;
    lista->inicio = nuevoNodo;
}
```

- `mostrarLista`: Imprime el contenido de la lista.

```
void mostrarLista(struct Lista lista) {
    if (lista.inicio == NULL) {
        printf("La lista está vacía.\\n");
        return;
    }
    printf("Contenido de la lista:\\n");
    struct Nodo* actual = lista.inicio;
    while (actual != NULL) {
        // ... (imprimir los datos de la persona en el nodo)
        actual = actual->siguiente;
    }
}
```

- `eliminarLista`: Elimina todos los nodos de la lista.

```
void eliminarLista(struct Lista* lista) {
    while (lista->inicio != NULL) {
        struct Nodo* temp = lista->inicio;
        lista->inicio = temp->siguiente;
        free(temp);
    }
}
```

Menú de Listas (`menuListas`):
El menú para las listas utiliza estas funciones para operaciones como agregar personas al inicio, mostrar la lista y eliminar la lista completa. Los mensajes guían al usuario a través de estas operaciones.

4. Árboles

Un árbol binario de búsqueda se utiliza para almacenar información de personas ordenadas por sus nombres. Las funciones permiten la manipulación y visualización del árbol.

Estructuras Relacionadas con Árboles:

```
struct NodoArbol {
    struct Persona datos;
    struct NodoArbol* izquierda;
    struct NodoArbol* derecha;
};

struct Arbol {
    struct NodoArbol* raiz;
};
```

- *Funciones Relacionadas con Árboles

:**

- `crearNodoArbol`: Crea un nuevo nodo para el árbol.

```
struct NodoArbol* crearNodoArbol(struct Persona datos) {
    struct NodoArbol* nuevoNodo = (struct NodoArbol*)malloc(s
izeof(struct NodoArbol));
    // ... (verificación de la asignación de memoria)
    nuevoNodo->datos = datos;
    nuevoNodo->izquierda = NULL;
    nuevoNodo->derecha = NULL;
    return nuevoNodo;
}
```

- `insertarEnArbol`: Inserta una nueva persona en el árbol.

```
void insertarEnArbol(struct NodoArbol** raiz, struct Persona
datos) {
    if (*raiz == NULL) {
        *raiz = crearNodoArbol(datos);
    } else if (strcmp(datos.nombre, (*raiz)->datos.nombre) <
0) {
        insertarEnArbol(&(*raiz)->izquierda, datos);
    } else {
        insertarEnArbol(&(*raiz)->derecha, datos);
    }
}
```

- `mostrarArbolInOrden`: Muestra el árbol en orden.

```
void mostrarArbolInOrden(struct NodoArbol* raiz) {
    if (raiz != NULL) {
        mostrarArbolInOrden(raiz->izquierda);
        // ... (imprimir los datos de la persona en el nodo)
        mostrarArbolInOrden(raiz->derecha);
    }
}
```

Menú de Árboles (`menuArboles`):
El menú específico para los árboles utiliza estas funciones para agregar personas al árbol y mostrar el árbol en orden. Los mensajes en el menú guían al usuario a través de estas operaciones.

5. Ordenamiento por Inserción

El ordenamiento por inserción se implementa para ordenar un arreglo de estructuras `Persona` por sus nombres.

Funciones Relacionadas con el Ordenamiento:

- `ordenamientoInsercion`: Realiza el ordenamiento por inserción.

```
void ordenamientoInsercion(struct Persona arreglo[], int n) {
    for (int i = 1; i < n; i++) {
        struct Persona key = arreglo[i];
        int j = i - 1;
        while (j >= 0 && strcmp(arreglo[j].nombre, key.nombr
e) > 0) {
            arreglo[j + 1] = arreglo[j];
            j = j - 1;
        }
        arreglo[j + 1] = key;
    }
}
```

- `mostrarArreglo`: Muestra el contenido de un arreglo de estructuras `Persona`.

```
void mostrarArreglo(struct Persona arreglo[], int n) {
    for (int i = 0; i < n; i++) {
        // ... (imprimir los datos de la persona en el arregl
o)
    }
}
```

Menú de Ordenamiento (`menuOrdenamientoInsercion`):
El menú para el ordenamiento por inserción permite al usuario ingresar varias personas, ordenar las personas por sus nombres y mostrar el arreglo ordenado.

Función Principal (`main`)

La función principal del programa (`main`) muestra un menú principal que permite al usuario seleccionar entre trabajar con pilas, colas, listas, árboles o el método de ordenamiento por inserción. Cada opción del menú principal llama a la función de menú correspondiente para realizar las operaciones deseadas.

Menú Principal:

```
int main() {
    int opcion;
    do {
        printf("\\n--- Menú Principal ---\\n");
        printf("1. Pilas\\n");
        printf("2. Colas\\n");
        printf("3. Listas\\n");
        printf("4. Árboles\\n");
        printf("5. Ordenamiento por Inserción\\n");
        printf("6. Salir\\n");
        printf("Ingrese una opción: ");
        scanf("%d", &opcion);
        switch (opcion) {
            case 1:
                menuPilas();
                break;
            case 2:
                menuColas();
                break;
            case 3:
                menuListas();
                break;
            case 4:
                menuArboles();
```

```
                break;
            case 5:
                menuOrdenamientoInsercion();
                break;
            case 6:
                printf("Saliendo...\\n");
                break;
            default:
                printf("Opción inválida. Intente de nuev
o.\\n");
        }
    } while (opcion != 6);
    return 0;
}
```

El menú principal se repite hasta que el usuario selecciona la opción de salir (`6`). Las opciones del 1 al 5 llaman a los menús específicos para pilas, colas, listas, árboles y el método de ordenamiento por inserción, respectivamente.

Espero que esta explicación detallada del código y las funciones te sea útil. ¡Si tienes alguna pregunta o necesitas más información, házmelo saber!